VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

# AUTENTIZACE UŽIVATELŮ WEBOVÝCH SLUŽEB

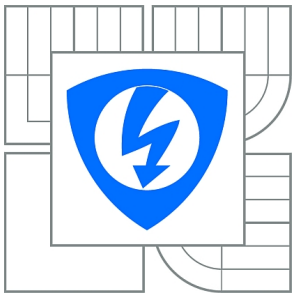
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
MATOUŠ PLAŠIL

BRNO 2013


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY


FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS


# AUTENTIZACE UŽIVATELŮ WEBOVÝCH SLUŽEB

AUTHENTICATION OF WEB SERVICE USERS

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS


AUTOR PRÁCE — MATOUŠ PLAŠIL

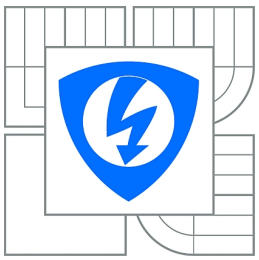
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. KAREL BURDA, CSc.

SUPERVISOR



BRNO 2013



**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**

**Fakulta elektrotechniky a komunikačních technologií**

**Ústav telekomunikací**

# Bakalářská práce

bakalářský studijní obor
**Teleinformatika**

| | | | |
|---|---|---|---|
| ***Student:*** | Matouš Plašil | ***ID:*** | 134383 |
| ***Ročník:*** | 3 | ***Akademický rok:*** | 2012/2013 |

**NÁZEV TÉMATU:**

## Autentizace uživatelů webových služeb

**POKYNY PRO VYPRACOVÁNÍ:**

Úkolem práce je popsat systémy jednotného přihlášení (SSO) používané pro přístup k webovým aplikacím. Student by měl popsat funkci takovýchto služeb z pohledu klientské webové aplikace. Praktickou částí práce je návrh webové aplikace, kde si uživatel bude moci zvolit minimálně dva způsoby přihlášení (Google ID, Live ID, ...) a nebo přihlášení přes lokální účet.

**DOPORUČENÁ LITERATURA:**

[1] SURHORE, Lambert, TENNOE, Mariam, HENSSONOW Susan. Windows Server Domain. Betascript Publishing, 2010. 174s. ISBN: 978-6132266194
[2] PROSISE, Jeff. Programování v Microsoft .NET, Webové aplikace v C#, ASP.NET a .NET Framework. Computer Press, 2003. 736s. ISBN: 80-7226-879-1

| | | | |
|---|---|---|---|
| ***Termín zadání:*** | 11.2.2013 | ***Termín odevzdání:*** | 5.6.2013 |

***Vedoucí práce:*** doc. Ing. Karel Burda, CSc.
***Konzultanti bakalářské práce:***

**prof. Ing. Kamil Vrba, CSc.**
*Předseda oborové rady*

**UPOZORNĚNÍ:**

## ABSTRAKT

Práce popisuje používané metody pro Single Sign-On z pohledu klientské aplikace. Jsou zde podrobně rozebrány protokoly OpenID a OAuth a jejich odlišnosti. Další část práce se zabývá popisem registrace aplikace a komunikací serveru s klientskou aplikací pomocí protokolu OAuth. V praktické části jsou popsány a vysvětleny stěžejní funkce webové aplikace.

## KLÍČOVÁ SLOVA

autentizace, autorizace, Single Sign-On, OpenID, OAuth, ASP.NET

## ABSTRACT

In this bachelor thesis are presented methods for Single Sign-On from perspective of client application. It takes a closer look at protocols OpenID and OAuth and explains the main differencies. Next part of the thesis is focused on registration of application and communication between server and client application. This is realized by OAuth protocol. In the practical part is described a created web application and used methods are explained.

## KEYWORDS

authentication, authorization, Single Sign-On, OpenID, OAuth, ASP.NET

PLAŠIL, Matouš *Autentizace uživatelů webových služeb*: bakalářská práce. Místo: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, Rok. 52 s. Vedoucí práce byl doc. Ing. Karel Burda, CSc.

# PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Autentizace uživatelů webových služeb“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Místo ..... .....

(podpis autora)

# PODĚKOVÁNÍ

Rád bych poděkoval panu Ing. Ondřejovi Morskému za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.


Místo ..... .....

(podpis autora)

# OBSAH

# SEZNAM OBRÁZKŮ

# ÚVOD

V současné době se na internetu vyskytuje velké množství služeb, které vyžadují ověření totožnosti uživatele. Většina stávajících služeb vyžaduje při prvním použití registraci, na základě které si uloží do databáze přihlašovací údaje pro danou fyzickou osobu. Při dalším používání si služba ověřuje zadané informace od fyzické osoby s přihlašovacími údaji ve své databázi a tímto procesem je uživatel autentizován. Tento způsob řešení však není zcela ideální. Uživatel je nucen provádět registraci pro každou novou službu, kterou chce používat. Musí si pamatovat přihlašovací údaje pro každou službu zvlášť a vzniká zde riziko ztráty identity u špatně zabezpečených webových služeb.

Pro usnadnění přístupu ke službám a zvýšení bezpečnosti vznikly systémy jednotného přihlášení - Single Sign-On (SSO). SSO systémy umožňují jednotné přihlašování pro více různých služeb. Tato práce výše uvedené systémy popisuje a zároveň hodnotí vhodnost jejich použití pro webové aplikace. Zabývá se zejména v současnosti nejpoužívanějšími protokoly, a to OAuth a OpenID.

Praktická část obsahuje webovou aplikaci ASP.NET využívajících jednoho z těchto protokolů pro přihlášení skrze služby Google ID a Windows Live ID. Dále je možné vytvořit lokální účet a propojit ho s výše uvedenými službami.

# 1 ZÁKLADNÍ POJMY

## 1.1 Autentizace

Autentizace zajišťuje ověření identity uživatele v určitém systému. Existují tři základní způsoby jak identitu ověřit. Ověření identity může být na základě něčeho, co uživatel zná, co má nebo čím je [1].

V případě zabezpečení na základě znalosti (co uživatel zná) je nejčastější heslo. Heslo je řetězec znaků, pomocí kterého se uživatel autentizuje. Bezpečnost hesla závisí na jeho délce a použitých znacích (v dnešní době je obvyklá délka 6-10 znaků). Dostatečně složité heslo by mělo být odolné proti slovníkovému útoku, či útoku hrubou silou. Autentizace pomocí hesla probíhá tak, že uživatel předloží systému uživatelské jméno společně s heslem. Tyto informace si systém ověří, a pokud je ověření úspěšné, je uživatel autorizován. Výhoda hesla spočívá v jednoduché přenositelnosti a není třeba složitého autentizačního systému. Nevýhodou je, že heslo nemůže být moc složité, protože by si ho uživatel špatně pamatoval. Je zde také velké riziko vyzrazení hesla, což má za následek ohrožení bezpečnosti. Autentizace na základě znalosti je jedním z nejrozšířenějších způsobů. Tento způsob byl využit i v praktické části při tvorbě webové aplikace.

Druhým způsobem ověření identity je ověření na základě vlastnictví. Pro to, aby se uživatel mohl přihlásit, musí vlastnit nějaký fyzický objekt, často označovaný jako token. Jedná se o přenosná zařízení obsahující tajné informace, jako je složité heslo nebo kryptografický klíč. Tokeny mohou též provádět kryptografické výpočty a být realizovány buď pomocí karty, autentizačního kalkulátoru či USB tokenu. USB tokeny jsou v poslední době poměrně rozšířené. Pokud jde o specializované USB tokeny s vysokou bezpečností, tak jsou realizovány stejnou technologií jako čipové karty [1]. Výhodou tokenu oproti heslu je, že může obsahovat složité heslo či systém zpracovávající určité informace. Zde odpadá nutnost uživatele pamatovat si velmi složité heslo. Je velmi jednoduché zjistit odcizení tokenu a velmi obtížné token zkopírovat. Nevýhodou je, že nemusí být vždy kompatibilní se všemi zařízeními a v případě ztráty není možné se autentizovat [2].

Poslední metodou je metoda autentizace na základě biometrických informací. Systém autentizuje uživatele na základě kvantifikovatelné fyziologické či behaviorální informace. Nejčastější ověření je na základě otisku prstu, oční sítnice či proporce obličeje [1]. Velkou výhodou je praktická nemožnost ztratit tento typ informace a velmi obtížné zkopírování. Nevýhodou je složitá měřitelnost této informace. Biometrické systémy pracují s určitou tolerancí, která může vést k bezpečnostním rizikům. Tato metoda je nejsložitější a také nejnákladnější.

Pro zvýšení bezpečnosti se používá kombinace těchto tří metod. Takový způsob nazýváme vícefaktorová autentizace a jejím základem je použití alespoň dvou odlišných metod. V případě dvou hovoříme o dvoufaktorové autentizaci. Tou může být například platba pomocí karty u platebního terminálu. Pro to, abychom mohli zaplatit, potřebujeme mít fyzicky kreditní kartu (token) a musíme k ní znát příslušný PIN (heslo) [2].

## 1.2 Autorizace

Autorizace navazuje na Autentizaci. V momentě, kdy je uživatel autentizován, je nutné mu z hlediska bezpečnosti přidělit oprávnění. Autorizace tento problém řeší. Autorizaci můžeme charakterizovat jako proces přiřazení přístupových práv uživateli. Uživatelé mohou mít práva nastavena samostatně, či mohou patřit do skupin s předdefinovanými právy.[2]

# 2 POPIS SINGLE SIGN-ON PROTOKOLŮ

V současné době jsou nejrozšířenějšími protokoly OpenID a OAuth. Tyto protokoly jsou zároveň nejvíce podporovány v prostředí internetu službami (Google, Microsoft, Facebook, Twitter apod). Protokoly jsou níže popsány a je vysvětlena jejich vzájemná odlišnost při praktickém použití.

## 2.1 OpenID

OpenID je otevřený standard pro bezpečnou autentizaci uživatelů v prostředí internetu. Pro správné pochopení principu je však důležité definovat následující základní pojmy:

**Uživatel** je fyzická osoba využívající aplikaci, která potřebuje ověřit svojí identitu.
**Klientská aplikace** je aplikace, kterou uživatel využívá.
**Provider** je server zajišťující ověření identity uživatele.

Bezpečnou autentizací je míněno takové ověření, při kterém klientská aplikace nedostane uživatelovo heslo. Dostane pouze potvrzení od providera, zda byl uživatel úspěšně autentizovaný, či nikoli. Na základě nastavení providera také aplikace dostane základní informace o uživateli, jako je např. jméno nebo e-mailová adresa.

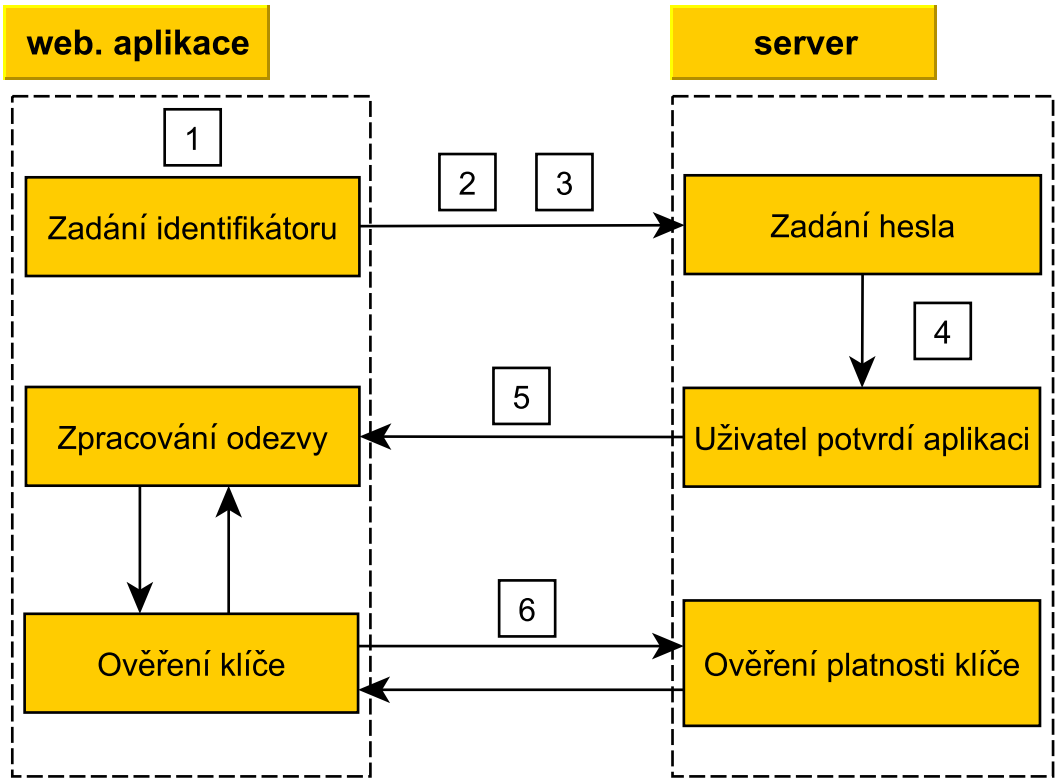
OpenID je decentralizovaný způsob autentizace. To znamená, že neexistuje jeden centrální provider, který by ověřoval všechny uživatele. Naopak providerů existuje velké množství a v principu se mezi sebou liší dvěma aspekty [7].

Prvním aspektem je druh zabezpečení. Většina OpenID providerů poskytuje jednofaktorové ověření identity (což je ve většině případů heslo). Někteří provideři však podporují vícefaktorové ověření.

Druhým aspektem je důvěryhodnost providera. Je důležité si uvědomit, že provider bude mít uživatelovy identifikační údaje, tudíž v případě špatného zabezpečení providera mohou být údaje odcizeny. Dalším problémem mohou být výpadky providera. Pokud nebude fungovat server providera, nebude se moci uživatel pomocí providera autentizovat. Proto je vhodné volit známé a důvěryhodné providery. Současnými poskytovateli jsou např. Google, AOL, Flickr, VeriSign (podpora vícefaktorové autentizace) a v České republice Seznam.

Aby se uživatel mohl přihlašovat pomocí OpenID, je třeba vybrat jednoho z providerů a provést u něj registraci. Na základě registrace bude uživateli vytvořen identifikátor. Identifikátor poté slouží pro přihlašování přes klientské aplikace a má tvar např. `novak.id.seznam.cz`. Po registraci uživatele u providera je možné přihlašování na klientských aplikacích, které podporují OpenID [7].

Mechanismus samotné autentizace je zobrazený na následujícím obrázku 2.1.




Obr. 2.1: Schéma OpenID autentizace

1. Uživatel ve webové aplikaci zadá do požadovaného formuláře svůj OpenID identifikátor.
2. Aplikace vyhledá providera k danému identifikátoru (nalezení je provedeno pomocí protokolu Yadis). Po nalezení stanoví koncovou URL adresu poskytovatele.
3. Aplikace přesměruje webový prohlížeč na autentizační server poskytovatele společně s autentizačním dotazem. Ten je proveden jako HTTP dotaz (GET či POST) a musí obsahovat následující parametry.

   **openid.ns** - tímto parametrem je určena použitá verze OpenID. V případě OpenID 2.0 má parametr hodnotu:
   `http://specs.openid.net/auth/2.0`.

   **openid.mode** - parametr pro určení druhu zprávy. Pokud tento parametr chybí, tak server předpokládá, že nejde o OpenID zprávu.
4. Uživatel na stránce providera napíše své heslo a potvrdí souhlas s autentizací pro danou aplikaci.

5. Server přesměruje prohlížeč zpět na webovou stránku společně s potvrzením, nebo zamítnutím autorizace. V případě potvrzení obsahuje odpověď následující parametry:
   **openid.ns**
   **openid.mode** - pole má hodnotu `ID_RE`.
   **openid.op_endpoint** - URL koncového bodu (autentizačního serveru)
   **openid.return_to** - kopie návratové URL poslána v requestu
   **response_nonce** - string o délce 255 znaků či kratší. Musí být unikátní. Na jeho začátku se nachází čas serveru, kdy byla odpoveď zpracována př. `2005-05-15T17:11:51ZUNIQUE`.
   **openid.assoc_handl**
   **openid.signed**
   **openid.sig**
   Pokud server autentizaci zamítl, tak odešle `checkID_immediate`. V takovém případě by měla web. aplikace vygenerovat nový dotaz s použitím parametru `checkID_SETUP`.
6. Webová aplikace provede ověřením, že informace o autentizaci je platná. Ověřuje se *Return URL*, parametr *nonc* a *signature*, což je sdílený klíč mezi webovou aplikací a autentizačním serverem, který byl vygenerován aplikací v momentě posílání requestu na server [7].

## 2.2 OAuth

OAuth je otevřeným standardem pro autentizaci uživatele a autorizaci aplikace v síti internet.

Existují dvě metody jak autentizaci pomocí OAuth realizovat. Těmi jsou **Implicit grant flow** a **Authorization code grant flow** [8]. Metody se mezi sebou liší průběhem autentizace. Implicit grant flow je jednodušší varianta autentizace, to však na úkor bezpečnosti a hodí se pro realizaci na jednoduchých zařízeních. **Authorization code grant flow** je složitější způsob autentizace.

V případě realizace webové aplikace je vhodnější využít metodu **Authorization code grant flow**, která je v následujícím textu popsána. Pro správné pochopení principu je opět jako u OpenID nutná definice základních pojmů.

**Uživatel** je fyzická osoba využívající aplikaci, která potřebuje ověřit svou identitu.

**Klientská aplikace** je aplikace, kterou uživatel využívá.

**Zdrojový server** je server, na kterém jsou uloženy informace, ke kterým chce klient přistupovat.

**Autorizační server** je server, který přiděluje přístupové tokeny po úspěšné autentizaci uživatele.

V některých případech může být **Zdrojový server** a **Autorizační server** spojen v server jeden [8]. Pro snažší vysvětlení bude v následujícím texu uvažováno, že jsou servery spojeny a budou nazývány obecným názvem **Provider**.

U strandardu OAuth je klientská aplikace svázána s providerem. Aby aby mohl klient poskytnout přihlášení přes určitého providera, musí být u providera registrován. Při registraci klient obdrží `client_id` a `secret` [8].

Během přihlašování přes OAuth je uživatel, stejně jako u OpenID, přesměrován na stránku providera, kde vloží své přihlašovací údaje a je ověřena jeho identita. Oproti OpenID je však dotázán na povolení přístupu aplikace k určitým zdrojům informací (autorizace aplikace) providera.

Klientská aplikace má vzhledem ke svým potřebám nastavené, jak velké oprávnění bude dotazovat. Uživatel může schválit přístup aplikaci, nebo zamítnout. Toto zabezpečení je velice důležité, protože nebýt autorizace aplikace, měla by klientská aplikace úplná práva uživatele.

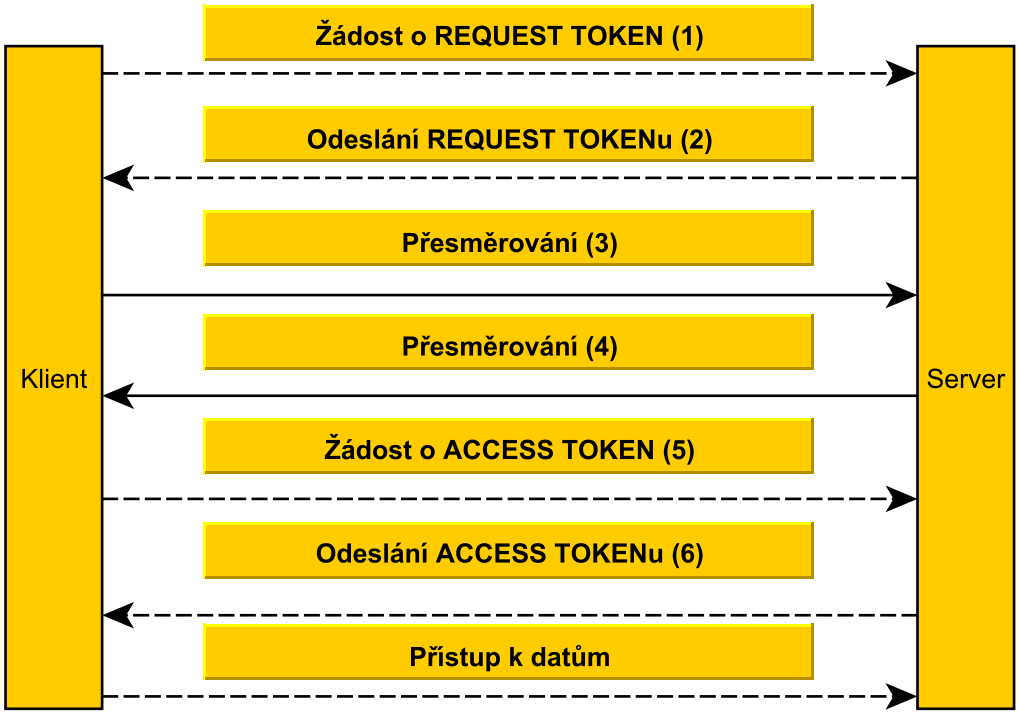
Po úspěšném odsouhlasení aplikace uživatelem proběhne výměna `client_id`, `secret` a autorizačního kódu. Poté získá klientská aplikace přístupový token. Pomocí přístupového tokenu může aplikace žádat požadované informace a data uživatele skrze API providera. Přístupový token má omezenou dobu platnosti.

Příklad použití OAuth: uživatel má účet na službě Google, kde má svá fotoalba, kontakty a osobní emaily. Chce použít online službu pro objednání tisku fotografií. Online služba nabízí přihlášení pomocí služby Google pro získání fotografií z fotoalba. Uživatel během přihlašování vidí, zda si aplikace žádá přístup pouze k fotoalbu, nebo mnohem více informací. Po odsouhlasení má aplikace možnost získat např. pouze fotky z fotoalba a uživatel nebude ohrožen tím, že bude aplikace přistupovat ke kontaktům nebo e-mailu.

Bezpečnostním problémem u tohoto standardu může být odcizení identifikačních údajů `secret` a `client_id`. Pokud tak nastane, útočník se může vydávat za danou aplikaci a zneužívat úroveň oprávnění poskytnuté aplikaci. Proto je důležité při výměně této informace s providerem zvolit bezpečnou metodu.

Autentizace pomocí OAuth probíhá dle obrázku 2.2 v následujících krocích:

1. Klient žádá server o přidělení **request tokenu**, posílá mu své **client ID**, čímž se identifikuje,
2. Server vygeneruje náhodný **request token** a zasílá ho zpět klientovi,
3. V této chvíli uživatel klikne na tlačítko, či odkaz, který provede přesměrování na server s danými parametry. Uživateli se zobrazuje přihlašovací obrazovka služby, a po přihlášení je vyzván ke schválení přístupu aplikace,
4. Po schválení je uživatel přesměrován na stránku klienta,
5. V tomto momentě zasílá klient žádost o **access token**,
6. Server posílá klientovi **access token**,
7. Nyní může přistupovat klient k API serveru a získávat informace [8].




Obr. 2.2: Obecný model funkce OAuth

## 2.3 Porovnání a výběr vhodného protokolu

OpenID je vhodný pro univerzální autentizaci uživatele. Jeho hlavní výhodou je velká škála providerů, ze kterých si uživatel může vybrat. Výhodou je také vyhledání providera klientskou aplikací pomocí uživatelského identifikátoru (tuto funkci má na starost protokol Yadis).

Oproti tomu OAuth plní jak autentizační funkci, tak i autorizační. Autorizační část je v současné době velmi potřebná, jelikož je třeba omezit aplikaci přístup k datům jen nezbytně nutným. Toto je hlavní rozdíl mezi OpenID a OAuth - nastavení práv aplikaci. Uživatelé mají v současné době velké množství dat uložené na službách jako je Google, Microsoft či Facebook. Je tedy nezbytné, aby byly přistupující aplikace omezeny.

Díky možnosti autorizace je OAuth perspektivnějším protokolem při použití klientské aplikace. Tento protokol bude také využit při realizace praktické části.

# 3 ZPŮSOBY KOMUNIKACE MEZI KLIENTEM A PROVIDEREM

## 3.1 URI

Jednotný identifikátor zdroje - Uniform Resource Identifier (URI) se využívá pro jednotnou identifikaci cílové informace v počítačové síti či síti internet. URI je složen z posloupnosti znaků, které dodržují základní syntax předepsanou dle RFC 3986 [6].

URI slouží pro popis hierarchické struktury nebo unikátního názvu, může však obě tyto informace obsahovat zároveň. Do URI spadá jednotný identifikátor cesty - Uniform Resource Locator (URL) a jednotný identifikátor názvu - Uniform Resource Name (URN) [5].

Obecné URI se skládá z těchto částí:

```
Schéma ":" hierarchická struktura [ "?" dotaz ] [ "#" fragment ]
```

**Schéma** je na začátku každého URI, skládá se z písmen číslic či znaků `"+"`, `","`, `"."`, `"-"` a ukončuje se znakem `":"`. Slouží k bližšímu určení použité syntaxe. Při zobecnění lze říci, že schéma určuje typ použitého protokolu.Příklady známých schémat: `"http:"` a `"ftp:"`.

**Hierarchická struktura** určuje hierarchickou cestu k požadované informaci. U relativní nebo absolutní cesty se začíná znakem `"/"`. Při určování cesty k určité autoritě začíná dvěma lomítky `"//"`. Příklad určení autority `"//google.com/"`.

**Dotaz** obsahuje data, která mohou být vstupními daty pro skript. Dotaz začíná znakem `"?"` a ukončen je znakem `"#"`. Jednotlivé dotazy jsou poté vkládány syntaxí `"klíč=hodnota"`. V případě více dotazů se používá oddělovací znak `"&"`. Příklad: `"?jmeno=Jan&vek=22#"`.

**Fragment** umožňuje nepřímou identifikaci sekundární informace pomocí odkazu na informaci původní. V praxi se fragment používá jako „záložka“ v rámci načtené stránky [6].

## 3.2 Metody GET a POST

Hypertextový protokol - Hypertext Transfer Protocol (HTTP) používá dva rozdílné druhy metod pro zasílání a přijímání informací. Těmito metodami jsou GET a POST. Metody se mezi sebou liší převážně ve způsobu kódování dat [3].

### 3.2.1 GET

Primární funkcí této metody je získání informací od serveru. Princip činnosti je takový, že data potřebná k odeslání se připojí za požadovanou URI adresu. To znamená, že data jsou odeslána jako dotazy (query strings). Metoda GET je velice jednoduchá pro realizaci, ale má řadu omezení a bezpečnostních problémů. Jedním z problémů je omezená délka URI, dalším může být omezení pouze na ASCII kódovou tabulku. Hlavním bezpečnostním problémem je transparentnost odesílaných dat. Jelikož jsou data odesílána pomocí URI, budou uložena v historii prohlížeče, mohou být uložena v cache a během průchodu sítí jsou přístupná ostatním zařízením. Vzhledem k těmto skutečnostem není vhodné pomocí metody GET odesílat informace, které jsou citlivé (hesla, osobní informace z formulářů). Při jednoduchosti modifikace odesílaných dat se také nedoporučuje používat metodu GET pro přístup k systémům, které by mohla změněná vstupní proměnná ovlivnit (např. požadavek na databázi) [3] [4].

### 3.2.2 POST

V současné době se metoda POST používá pro odesílání dat na server. POST na rozdíl od metody GET neodesílá data pomocí URI adresy, ale data vkládá přímo do těla zprávy. Data se tedy neukládají do webového prohlížeče a nejsou viditelná uživatelem. POST se tedy používá pro odesílání dat, vyplněných formulářů, hesel apod. Výhodou oproti metodě GET je také to, že velikost dat není limitována maximální délkou URI a je možné odesílat i jiné znaky, než jsou obsaženy v tabulce ASCII [3] [4].

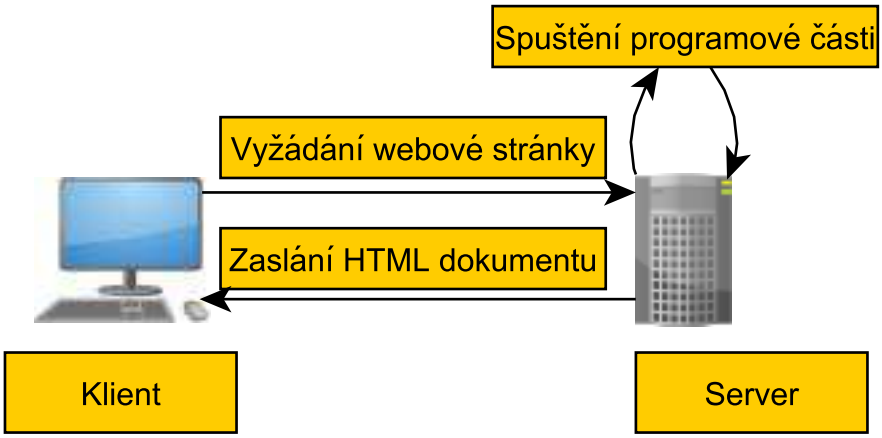
# 4 POPIS POUŽITÉ TECHNOLOGIE

## 4.1 ASP.NET

V počátcích vzniku webových stránek a HTML se jednalo vetšinou o statické stránky a nikoli o webové aplikace, které by nějakým způsobem reagovaly na to, co uživatel na stránce dělá. Webový prohlížeč pouze zaslal požadavek na určitý server a ten mu odeslal statický kód ve formátu HTML, který prohlížeč přeložil uživateli.

Tyto nedostatky začal řešit HTML ve verzi 2.0, který integroval první základy programování nazývané HTML *forms*. Obsahoval prvky jako jsou check boxy, rolovací menu, tlačítka aj. Po nastavení jednotlivých prvků uživatel svou volbu potvrdil a informace byly odeslané v podobě stringu na server, kde mohly být dále zpracovávány. Spousta ovládacích prvků z HTML *forms* se stále využívá i v ASP.NET. Rozdíl mezi ASP.NET a HTML *forms* je v typu aplikace, která běží na serveru [9].

Dynamické stránky lze rozdělit na ty, které zpracovávají programové části na serveru, a ty které jsou zpracovávány u klienta. ASP.NET je zkonstruováno pro běh programové části na straně serveru. Schéma průběhu lze vidět na obrázku 4.1




Obr. 4.1: Zpracování programu na straně serveru

ASP.NET je sada nástrojů od Microsoftu pro vývoj webových aplikací. Je součástí .NET frameworku. .NET framework je velice rozsáhlý a lze pomocí něj psát rozsáhlé aplikace a služby. Vývojář používající .NET framework si může vybrat z více programovacích jazyků, kterými jsou např. C# nebo Visual Basic.

Jedná se o vysokoúrovňovou vývojovou platformu, která dovoluje tvořit dynamické stránky bez starostí s implementací programu na nízkých úrovních. ASP.NET

obsahuje nástroje pro implementaci zabezpečení, práce s daty či ukládání uživatelských nastavení [9].

## 4.2 MVC

Jedná se o architekturu, která byla vytvořena pro přehlednější návrh ASP.NET aplikací. Architektura Model-View-Controller (MVC) dělí aplikaci na tři logické části. Tyto části lze upravovat samostatně a je snaha, aby se při vlastní změně co nejméně ovlivňovaly. Části modelu MVC jsou Model, View a Controller [10].

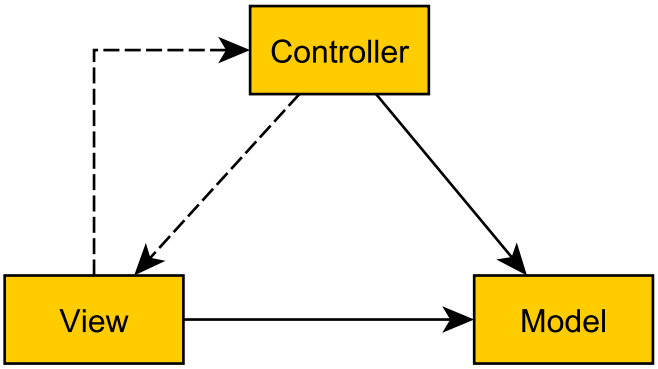
**Model:** v této části jsou uložena data aplikace. Model má také na starost business logiku, což je logika vztažená k datům, a tudíž logicky patří k modelu.

**View:** část view je v podstatě jediná část, kterou uživatel vidí. Jedná se o prezentační část programu, což je uživatelské rozhraní - User Interface (UI).

**Controller:** spouští procesy vyvolané nějakou událostí. Je to řídící logika aplikace.

Spojením těchto částí vzniká přímý odkaz Controlleru na Model. To zajišťuje možnost Controlleru upravovat data Modelu. View má přímý odkaz na Model, aby model mohl vykreslit data v něm uložená. Model je zobrazen na obrázku 4.2.

V modelu nesmí existovat přímá vazba od Modelu na ostatní dvě komponenty, to by byla zásadní chyba v návrhu aplikace. V některých schématech je možné nalézt „nepřímou vazbu“ Modelu na View. Nepřímá vazba slouží k notifikaci View při změně dat v Modelu. Část View je tedy informována o tom, aby vykreslila nová data. [10].




Obr. 4.2: Komunikace v architektuře MVC

## 4.3 IIS Express

Pro testování a spouštění webových aplikací v ASP.NET jsou dostupné servery ASP.NET vývojový server (obsažený ve vývojovém nástroji Visual Studio) nebo ISS webový server (dostupný v operačním systému Windows).

Obě řešení však mají své výhody i nevýhody. V případě ASP.NET serveru pro Visual Studio jde o zjednodušenou verzi serveru, která má sice snadnou instalaci a obsluhu, ale chybí jí některé funkce. Funkce jako je například možnost spuštění webové aplikace na vzdáleném portu či použití SSL.

ISS web server nabízí oproti ASP.NET serveru více funkcí, jako je právě SSL. Nevýhodou však je však nutnost administrátorského přístupu při ladění programu, nebo vzájemná nekompatibilita mezi verzemi. Obsluha a instalace ISS serveru je také náročnější.

Na základě těchto problémů vznikl server IIS Express, který kombinuje oba výše uvedené servery. Z obou serverů jsou vybrány lepší vlastnosti, jako je snadná správa a instalace při stejném množstvím funkcí jako u IIS Web Server [11].

Některé vlastnosti ISS Express serveru:

- pro ladění a běh aplikací není nutné administrátorské oprávnění
- obsahuje všechny funkce jako klasický ISS web server
- podporuje rozšiřující moduly a nastavení pomocí `web.config`
- může být nainstalován jak společně s ISS web serverem, tak i s ASP.NET serverem pro Visual Studio
- může běžet na Windows XP nebo novějším operačním systému

## 4.4 LocalDB

LocalDB slouží pro běh databáze a spadá pod SQL Express server. LocalDB bylo navrženo speciálně pro vývojáře a je velice snadné jej nainstalovat. Velkou výhodou je možnost běhu databáze bez nutnosti konfigurovat SQL Express server [12]. LocalDB je oproti SQL Express odlehčená verze, zabírá méně místa (velikost instalace 32 MB) a má stejné programovatelné funkce jako SQL Express (120 MB). Adresa serveru pro LocalDB je `(localdb)\v11.0`.

Hlavní rozdíly mezi LocalDB a SQL Server Express:

- LocalDB oproti SQL Server Express neběží jako služba, ale jako aplikace. Aplikace je navíc spouštěna na požádání
- LocalDB je možné provozovat pouze lokálně a není určen pro více uživatelů nebo pro nasazení na server
- V případě vzdáleného připojení nebo administrace je taktéž nutné použít SQL Server Expres [12]

Provozovat LocalDB je možné až od verze SQL Server 2012 Express LocalBD a ke konfiguraci databáze lze použít SQL Management Studio 2012. Při instalaci Visual Studio Express for Web 2012 je LocalDB automaticky nainstalována.

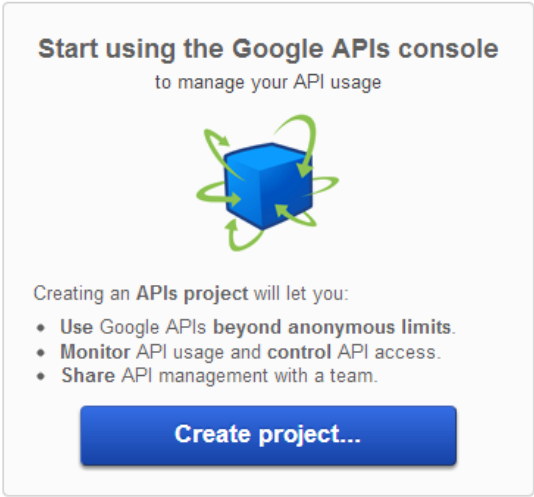
# 5 IMPLEMENTACE METOD U PROVIDERA

## 5.1 Registrace aplikace

### 5.1.1 Google

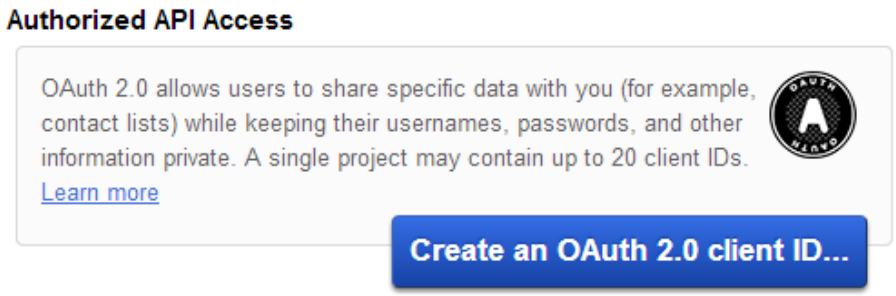
Pro zprovoznění protokolu OAuth je nutné zaregistrovat klientskou aplikaci. Pro registraci aplikace na službě Google APIs je nutné mít uživatelský účet Google, který je možné vytvořit na této adrese www.accounts.google.com. Registrace se poté provádí na této adrese www.accounts.google.com [13].

Po přihlášení do služby Google Apis se zobrazí možnost vytvoření nového projektu dle obrázku 5.1. Tuto volbu je nutné zvolit.




Obr. 5.1: Vytvoření nového projektu

Vytvořením nového projektu je možné vytvářet přístup k API Google. Nyní je třeba se přepnout na záložku **API Access**. Zde se zaregistruje klientská aplikace kliknutím na **Create an OAuth 2.0 client ID...** viz obrázek 5.2.




Obr. 5.2: Vytvoření client ID

Zobrazí se formulář Create Client ID. V tomto formuláři je nutné vyplnit **Product name**, zde je na místě zvolit vhodné jméno, jelikož se bude zobrazovat uživateli. Dále je možné zvolit **Product logo** a **Home Page URL**.

V následujícím kroku je zvolen typ aplikace. Jelikož aplikace pro testování OAuth bude webová aplikace ASP.NET, nechá se označená **Web application**. U pole **Your site or hostname** je třeba zvolit položku (more options). Je to nutné z důvodu nastavení **Redirect URI**, která je od Google defaultně na:

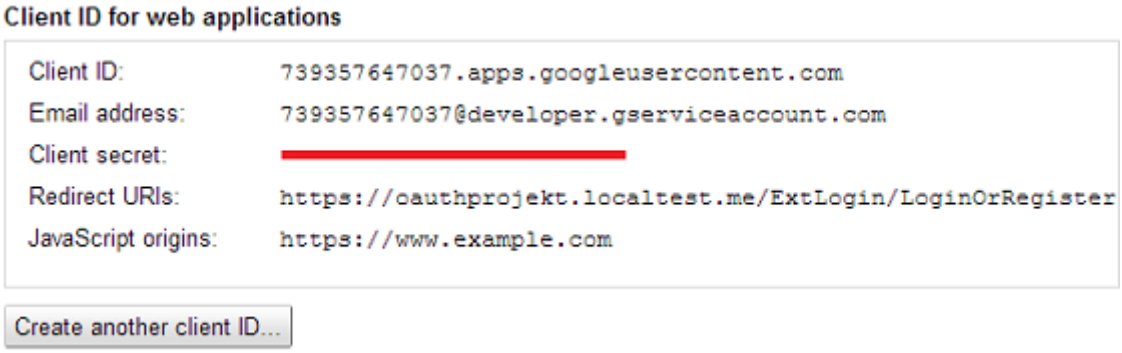
`https://www.example.com/oauth2callback`

Vytvořená aplikace však naslouchá na jiné adrese, tudíž pole **Authorized Redirect URIs** je třeba nastavit na:

`http://oauthprojekt.localtest.me/ExtLogin/LoginOrRegister`

Pole Authorized **JavaScript Origins** zůstane beze změny.

Po kliknutí na **Create Client ID**, je aplikace úspěšně zaregistrovaná. Je vygenerováno `Client ID`, což je identifikátor aplikace, `Email address` a `Client secret`. Příklad vygenerovaných identifikačních informací je na obrázku 5.3. `Client secret` je nutné udržet v tajnosti, jelikož se tímto heslem aplikace ověřuje. V případě odcizení tohoto hesla je možné se za danou aplikaci vydávat. Pod jedním uživatelským účtem je možné registrovat více aplikací.




Obr. 5.3: Zaregistrovaná aplikace Google

Tímto je registrace Google aplikace kompletní.

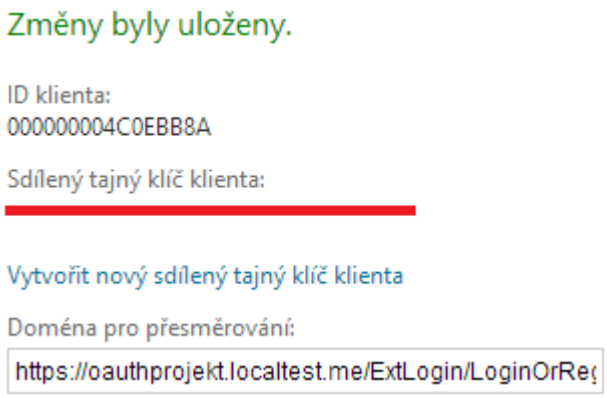
### 5.1.2 Microsoft

Pro registraci aplikace na službě Windows Live je nutné mít uživatelský účet Windows Live ID. Ten je možný vytvořit na adrese www.signup.live.com. Registrace aplikace se provádí na této adrese www.manage.dev.live.com.

Po přihlášení je třeba vybrat název aplikace (tento název se bude zobrazovat uživateli), vybrat jazyk aplikace a odsouhlasit podmínky používání služeb Live Connect. Po odsouhlasení je vytvořeno **ID klienta** a **Sdílený tajný klíč klienta**. Nyní je nutné nastavit **Doménu pro přesměrování**, jedná se o určení adresy, na kterou bude Microsoft odesílat odpověď aplikaci (klientovi). Adresu pro přesměrování je třeba nastavit na níže uvedenou hodnotu a uložit změny. Hodnota adresy pro přesměrování:

`http://oauthprojekt.localtest.me/ExtLogin/LoginOrRegister`

Tímto je registrace hotová. Výstup by měl být podobný obrázku 5.4.




Obr. 5.4: Registrovaná aplikace Microsoft

## 5.2 Komunikace - klient a provider

Tato část popisuje praktickou komunikaci mezi klientskou aplikací a providery (Google a Microsoft). U obou providerů byl zvolen způsob autentizace OAuth2.0 s metodou **Authorization code grant flow**. Tato metoda byla zvolena s ohledem na větší bezpečnost oproti metodě **Implicit grant flow**.

Komunikace mezi providerem a klientem začíná tak, že uživatel vybere volbu přihlášení přes providera. Po výběru je jeho prohlížeč přesměrován na URL providera s dotazovacími parametry, které vyžaduje providerovo API. Uživatel je ověřen zadáním uživatelského jména a hesla. Poté je nutné odsouhlasit přístup aplikaci.

Výsledkem tohoto procesu je získání autorizačního kódu. Autorizační kód je vrácen klientské aplikaci v podobě přiloženého dotazu v URL [14].

Po přijetí autorizačního kódu zažádá klientská aplikace o přístupový token. Aplikace odesílá své `client_id` a `client_secret` (získané během registrace aplikace) společně s autorizačním kódem. Po obdržení přístupového tokenu může aplikace přistupovat k API providera.

Pokud autorizační kód obsahuje obnovovací token, může být použit pro vyžádání nového přístupového tokenu v případě vypršení platnosti. Tento typ přístupu se nazývá offline, to znamená, že pro získání nového přístupového tokenu není potřeba, aby měl klient spuštěný webový prohlížeč [14].

V případě providera Google je žádost o autorizační kód odeslána na následující URL adresu:

`https://accounts.google.com/o/oauth2/auth`

Provider Microsoft používá pro žádost o autorizační kód tuto URL adresu:

`https://login.live.com/oauth20_authorize.srf`

Výše uvedená adresa je přístupná pouze skrze SSL a nešifrovaná spojení HTTP jsou zahozena. Za výše uvedenou URL adresou následují tyto dotazy (query strings):

**response_type** určuje formát odpovědi, kterou provider odešle zpět aplikaci. V případě webové aplikace se používá parametr `code`.

**client_id** je jedinečný identifikátor klientské aplikace získaný registrací.

**redirect_uri** určuje adresu, na kterou bude odeslána odpověď s autorizačním kódem. Hodnota této proměnné musí být shodná s hodnotou **redirect_uri** zadanou při registraci aplikace. V případě rozdílných **redirect_uri** bude vrácena chyba: `redirect_uri mismatch`.

**scope** slouží pro nastavení práv, které bude klientská aplikace vyžadovat. Například hodnota `https://www.googleapis.com/auth/userinfo.email` žádá v případě použití Google o název uživatelova e-mailu. Tuto volbu také využívá zrealizovaná aplikace. Větší výčet hodnot pro parametr **scope** lze nalézt pro Google na adrese www.developers.google.com. V případě providera Microsoft lze nalézt informace zde: www.msdn.microsoft.com

**state** slouží jako pomocná proměnná klientské aplikaci. State může obsahovat jakýkoli string. Po zpracování odpovědi je odeslán ten samý string do aplikace. Tento parametr je vhodný využít v případě, že klientská aplikace volá autentizaci z více míst a potřebuje identifikovat, z kterého místa byla autentizace zavolána.

**access_type** určuje, zda klientská aplikace potřebuje přistupovat k API providera i během chvíle , kdy uživatel nemá otevřený prohlížeč. Parametr je defaultně nastaven na hodnotu `online`. Pokud bude hodnota nastavena na `offline`, tak bude při první autentizaci uživatele zaslán klientské aplikaci i obnovovací token.

**approval_prompt** tímto parametrem se nastavuje, zda bude uživatel žádán pokaždé pro odsouhlasení aplikace při přihlašování, nebo zda bude dotázán pouze při prvním použití klientské aplikace. V případě Google je defaultně tento parametr nastaven na hodnotu `auto`, tedy uživatel schválí aplikaci pouze poprvé. V případě přiložené aplikace je parametr nastaven na `force`, tudíž je uživatel tázán při každém přihlášení. `Force` je nastaveno z důvodu přehlednějšího testování. Microsoft má defaultní volbu `force` [14].

**login_hint jen google** pokud má klientská aplikace informaci, který uživatel se chce autentizovat, může poskytnout nápovědu providerovi. Nápovědou je zde myšlena například e-mailová adresa. Odeslání této informace způsobí, že uživatel bude mít např. předvyplněné uživatelské jméno (např. e-mailovou adresu). Jedná se tedy o ulehčení autentizace uživatelovi a zrychlení celého autentizačního procesu [14].

Příklad finální žádosti o autorizační kód na providera Google:

```
https://accounts.google.com/o/oauth2/auth?
scope=https://www.googleapis.com/auth/userinfo.email
&redirect_uri=http://oauthprojekt.localtest.me/Login/LoginOrRegister
&response_type=code&client_id=739357647037.apps.googleusercontent.com
&approval_prompt=force
```

V momentě, kdy se uživatel přihlásí u providera a odsouhlasí aplikaci, je odeslán autorizační kód pomocí URI na zadanou adresu `redirect_uri`. Formát přijaté odpovědi v případě úspěšné autentizace může být následující:

```
http://oauthprojekt.localtest.me/Login/LoginOrRegister?
state=/profile&code=4/P7q7W91a-oMsCeLvIaQm6bTrgtp7
```

Pokud se autentizace nezdařila, je zaslána chybová odpověď uvedená níže.

```
http://oauthprojekt.localtest.me/Login/LoginOrRegister?
error=access_denied&state=/profile
```

Aplikace, která vlastní autorizační kód, může tento kód vyměnit za přístupový token. Tato žádost se provádí pomocí metody HTTPs POST. Samotná žádost musí obsahovat následující parametry [14]:

**code** autorizační kód přijatý z prvotní žádosti

**client_id** jednotný identifikátor klientské aplikace získaný po registraci u providera

**client_secret** heslo klientské aplikace získané registrací u providera

**redirect_uri** adresa pro odeslání odpovědi. Tato adresa musí být opět stejná jako adresa nastavená při registraci

Výsledná žádost poté může mít následující tvar:

```
POST /o/oauth2/token HTTP/1.1
Host: accounts.google.com
Content-Type: application/x-www-form-urlencoded
code=4/Pway91a-oMsCeLvjoaQm6bTrwdw8
&client_id=568757647037.apps.googleusercontent.com
&client_secret=client_secret
&redirect_uri=http://oauthprojekt.localtest.me/Login/LoginOrRegister
&grant_type=authorization_code
```

Úspěšná odpověď na tuto žádost bude poté obsahovat následující informace:

**access_token** přístupový token, který lze využívat pro přístup k API.

**refresh_token** jedná se o obnovovací token v případě, že při prvotní žádosti byl parametr **access_type** s hodnotou `offline`.

**expires_in** doba platnosti tokenu.

**token_type** určuje typ tokenu, který je navrácen. V případě webové aplikace bude vždy `Bearer` [14].

Přijatá odpověď je ve formátu pole JavaScript Object Notation (JSON) a pro práci s jednotlivými proměnnými je nutné toto pole deserializovat. Přijatá odpověď s přístupovým tokenem v případě Google může být následující:

```
"access_token":"1/fFAGRNJru1FTz70BzhT3Zg",
"expires_in":3920,
"token_type":"Bearer"
```

## 5.3 Přístup k API

Poté, co aplikace obdrží přístupový token, může přistupovat k informacím skrze API providera. Aplikace přikládá ke každému dotazu na API přístupový token, který je v proměnné `access_token`. Výsledný dotaz na providera Google pro zaslání informací o uživateli může vypadat následovně:

```
GET https://www.googleapis.com/oauth2/v1/userinfo?
access_token=1/fFBGRNJru1FQd44AzqT3Zg
```

V případě volání providera Microsoft:

```
GET https://apis.live.net/v5.0/me?access_token=abcdef12345
```

Zde je nutné si uvědomit, že přístupový token je vztažený na určitý druh žádostí. Nelze tedy provádět jakékoli volání API, nýbrž jen ta volání, ke kterým dal uživatel souhlas v momentě autentizace pomocí protokolu OAuth. Parametr, kterým se nastavovala úroveň požadovaného oprávnění, se nazývá `scope`.

# 6 POPIS PROGRAMU

V rámci praktické části bakalářské práce byla vytvořena webová aplikace. Primárním cílem byla realizace přihlášení uživatele pomocí externích účtů (Google, Microsoft). Toto přihlašování bylo úspěšně realizováno pomocí protokolu OAuth.

Dalším úkolem byla možnost vytvoření lokálního účtu, funkce byla také úspěšně realizována. Mezi další možnosti patří propojení lokálního účtu s externím účtem, možnost přidání lokálního hesla u účtu vytvořeného pomocí externího přihlášení.

Přidány byly taktéž funkce pro komfortnější ovládání aplikace, jako je indikace přihlášeného uživatele a možnost okamžitého odhlášení. Realizace aplikace tedy byla úspěšná a všechny požadované části jsou funkční. Byla ověřena funkčnost protokolu OAuth u ASP.NET aplikace.

V následujícím textu jsou popsány stěžejní části aplikace. Kompletní projekt se nachází na DVD v příloze.

## 6.1 Spuštění programu

Jak bylo zmíněno výše, součástí této bakalářské práce je webová aplikace. Aplikace byla vytvořena v prostředí Visual Studio Express for Web 2012 a byla otestována na operačních systémech Windows 7 a Windows 8. Pro úspěšné zprovoznění je nezbytné mít nainstalovené Vistual Studio Express for Web případně jinou vyšší verzi Visual Studia. Verze VS Express for Web je volně dostupná ke stažení na adrese www.microsoft.com/visualstudio. Dále je potřebná registrace u služeb Google a Microsoft, která je popsána v sekci 5.1.

Na přiloženém DVD ve složce **aplikace** se nachází kompletní projekt. K otevření projektu slouží soubor **OAuthProjekt.sln**.

*Aby bylo možné úspěšně provést nastavení, je nutné spustit projekt ve Visual Studiu jako správce.*

Po otevření projektu je nutné nastavit údaje získané registrací u Google a Microsoftu. První údaj, který je třeba nastavit, bude odkaz pro zažádání o token, zde se doplňuje klientské ID přidělené od jednotlivých providerů během registrace. ID se nastavuje v metodě **CheckExt**, která obsahuje proměnné `google` a `microsoft`. Do těchto proměnných zadáme místo hodnot `ID_klient` hodnotu přidělených ID od jednotlivých providerů. Metoda se nachází ve třídě **ExtLoginController**. Cesta: **Controllers/ExtLoginController.cs**. Výsledné nastavení by mělo vypadat obdobně, jako je na obrázku 6.1.

```
public void CheckExt(string service, string chose)
{
    //Nastevení klientských ID získaných registrací u providera OAuth
    var google = "250717078609.apps.googleusercontent.com";
    var microsoft = "00000000400DFA79";
```

Obr. 6.1: Příklad nastavení klientského ID

V dalším kroku nastavení je nutné zadat údaje **klient ID** a **secret** jednotlivých providerů do metody **ExtLogin**, nacházející se také ve třídě **ExtLoginController**. V metodě se vyskytují dvě proměnné - `google` a `microsoft`. Tyto proměnné vznikají novou instancí třídy **serviceParameters**, která má tři vstupní parametry. Těmito parametry jsou: Adresa providera, ID klienta, secret klienta. Adresa providera zůstane nezměněná, je však nutné zadat správné parametry ID klienta a secret klienta. Nahradíme tedy `ID_klient` a `SECRET_klient` požadovanými parametry u obou proměnných. Výsledné nastavení by mělo vypadat obdobně, jako na obrázku 6.2

```
public Tuple<string, string, string> ExtLogin(string code, string error)
{
var google = new serviceParameters("https://accounts.google.com/o/oauth2/token",
"250717078609.apps.googleusercontent.com","[redacted]");
var microsoft = new serviceParameters("https://login.live.com/oauth20_token.srf",
"00000000400DFA79", "[redacted]");
```

Obr. 6.2: Příklad nastavení klient ID a klient secret

Další částí zprovoznění programu je přidání portu pro běh IIS Expres serveru. V defaultním nastavení se při spuštění projektu vygeneruje port, na kterém bude aplikace lokálně spuštěna. Při registraci **Redirect URI** u providerů byla využita stránka pro přesměrování na localhost zvaná **localtest.me**, ta však odkazuje vždy na `127.0.0.1:80`. Je třeba přidat tuto adresu do konfigurace, aby mohla aplikace přijímat odpovědi od providerů.

Nastavení se provádí editací souboru **applicationhost.config**, který se nachází v `C:\Users\"uživatel"\My Documents\IISExpress\config`. V souboru se vyhledá část s nastavením požadované webové aplikace - v případě testovací aplikace je to `OAuthProjekt`. Zde se mezi tagy `<site name>` nachází nastavení aplikace. Uvnitř tagů `<bindings>` jsou nastaveny adresy, na kterých aplikace běží a také přijímá odpovědi. Uvnitř binding je třeba přidat další řádek a vložit následující parametry:

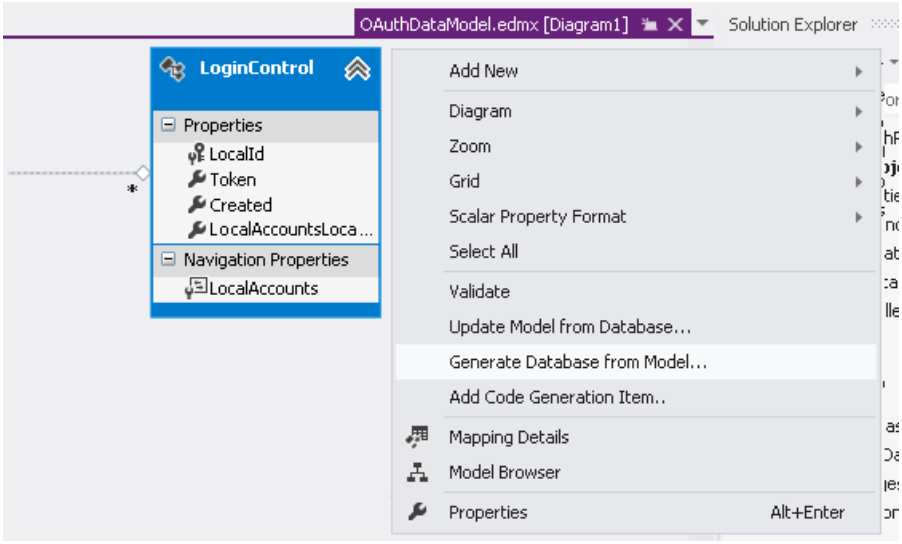
```
<binding protocol="http" bindingInformation="*:80:127.0.0.1" />
```

Výsledné nastavení by mělo být obdobné, jako je na následujícím obrázku 6.3. Provedené změny je třeba uložit.

```
<site name="OAuthProjekt" id="2">
<application path="/" applicationPool="Clr4IntegratedAppPool">
<virtualDirectory path="/"
 physicalPath="C:\Users\m0r4n_000\OAuthProjekt" />
</application>
<bindings>
   <binding protocol="http" bindingInformation="*:54490:localhost" />
   <binding protocol="http" bindingInformation="*:80:127.0.0.1" />
</bindings>
</site>
```

Obr. 6.3: Nastavení applicationhost.config v IIS

Posledním krokem ke zprovoznění programu je připojení vytvořené databáze. Nejprve je nutné otevřít ve Visual Studiu soubor **OAuthDataModel.edmx**. Jedná se o model databáze vytvořený pomocí Entity Framework. Po otevření je třeba kliknout pravým tlačítkem na pozadí pro otevření kontextového menu modelu a v zobrazené nabídce vybrat **Generate Database from Model** viz obrázek 6.4.

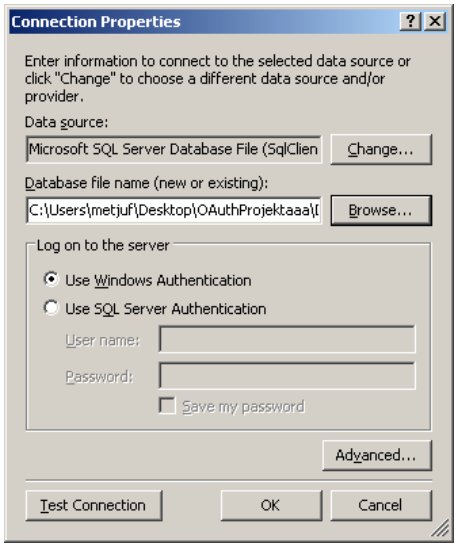



Obr. 6.4: Kontextová nabídka modelu

Zobrazí se průvodce pro generování databáze. V této části je třeba zvolit **New Connection...**

V nastavení připojení databáze je nutné zvolit cestu k existující databázi **Database file name (new or existing)**. Existující databáze **OAuthData.mdf** se nachází ve složce projektu v podsložce **DATA**. Zvolí se **OAuthData.mdf**. V této

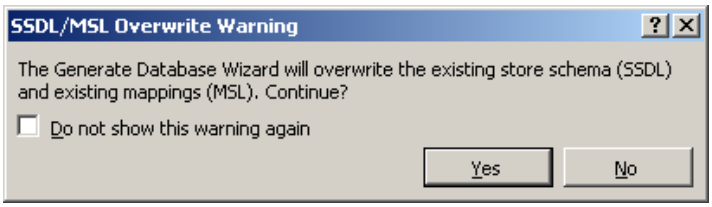
části je možné ověřit funkčnost databáze tlačítkem **Test Connection** a potvrdit nastavení **OK**. Nastavení je vidět na obrázku 6.5.




Obr. 6.5: Nastavení cesty k databázi OAuthData.mdf

Po potvrzení bude název nastavené databáze zobrazen v poli **Whitch data connection should your application use to connect to the database?** A detailní informace o propojení v poli **Entity connection string**.

Po kliknutí na tlačítko **next** se zobrazí vygenerovaný soubor `*.sql`. K dokončení připojení databáze je třeba zvolit **Finish**. Zobrazí se varování o přepsání původního schématu a předchozího mapování, jelikož již byla databáze namapována při předchozím testování viz obrázek 6.6. Zde je volba **Yes**. Nyní se spustí aplikace pomocí **Debug/Start Debugging**. V této části se zobrazí několikrát varování. Tyto varování je třeba potvrdit.




Obr. 6.6: Varování - přepsání SSDL

Po odsouhlasení varování se může zobrazit okno pro volbu databázového serveru, ke kterému se má projekt připojit. Zde je potřeba napsat cestu: `localdb\v11.0`.

V tomto okamžiku je nastavení kompletní. Spustí se webový prohlížeč s webovou aplikací a je možné aplikaci testovat.

Ačkoli se spustí prohlížeč s přiděleným portem, je vhodnější testovat webovou aplikaci pomocí stránky `http://oauthprojekt.localtest.me`.

*Pokud při pokusu spustit aplikaci vyskočí chyba se spuštěním server IIS Express (obrázek 6.7), nebylo Visual Studio puštěno v režimu správce. Je nutné ukončit Visual Studio a pustit ho znovu jako správce.*


Obr. 6.7: Chyba při spuštění aplikace

## 6.2 Databáze

Aplikace používá pro uchování registrovaných uživatelů databázi viz obrázek 6.8. Tato databáze je složena ze tří tabulek, kterými jsou **ExternalAccounts**, **LocalAccounts** a **LoginControl**.

Tabulka ExternalAccounts má uložené informace o registracích do webové aplikace, které proběhly pomocí externího přihlášení. Obsahuje následující položky:

**LocalId** jedná se o jednotný identifikátor uživatele v rámci aplikace, je zároveň klíčovou hodnotou tabulky. Má datový typ guid.

**Service** string, ve kterém je uložena informace, pomocí kterého providera byla registrace provedena. Může nabývat tedy pouze hodnot `google` nebo `microsoft`.

**ServiceId** je jedinečný identifikátor uživatele u providera. Tento údaj je aplikaci zaslán providerem během registrace.

Tabulka **LocalAccounts** slouží pro uložení lokální uživatelské přezdívky a hesla. Pokud se uživatel registroval pomocí externího providera, je převzata jeho přezdívka ze služby, pomocí které se přihlašoval, a položka heslo zůstane prázdná. Tabulka **LocalAccounts** obsahuje následující položky:

**LocalId** jedná se o jednotný identifikátor uživatele v rámci aplikace, je zároveň klíčovou hodnotou tabulky. Má datový typ guid.

**Email** uživatelské jméno pro přihlášení do webové aplikace. Datovým typem je string.

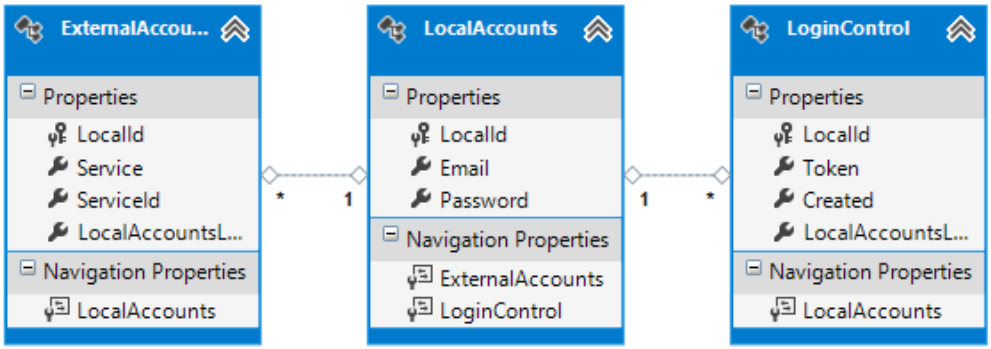
**Password** je zasolený haš hesla realizovaný pomocí algoritmu SHA1. V případě externího přihlášení není tato položka povinná. Datový typ string.

Tabulka **LoginControl** uchovává informace uživatelů, kteří jsou přihlášeni. Tabulka je kontrolována pomocí metody `Application_BeginRequest()` a v případě záznamu staršího než pět minut, je záznam vymazán. Uživatel je tedy přihlášen po dobu pěti minut, po uplynutí této doby se musí opět přihlásit.

**LocalId** jedná se o jednotný identifikátor uživatele v rámci aplikace, je zároveň klíčovou hodnotou tabulky. Má datový typ guid.

**Token** je náhodně vygenerovaný guid pro každé přihlášení.

**Created** čas vytvoření záznamu. Má datový formát Time.

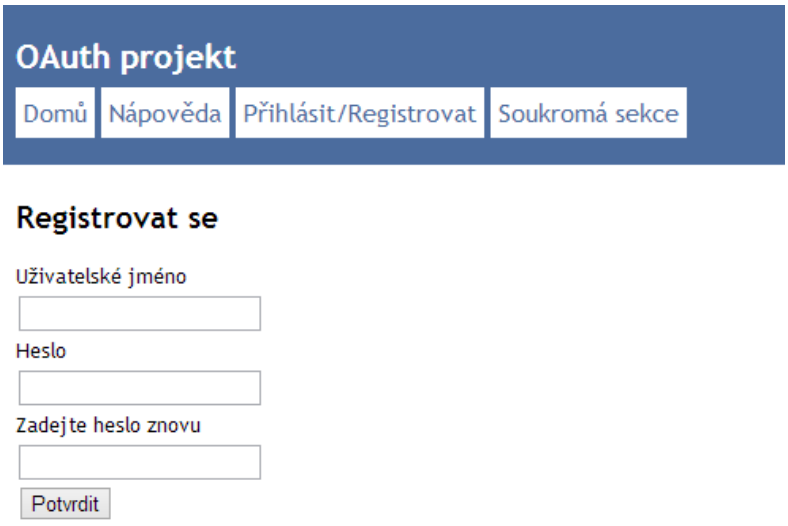



Obr. 6.8: Schéma databáze

## 6.3 Registrace a přihlášení lokálního uživatele

Program nabízí možnost lokální registrace uživatele. Vytvořený lokální účet je poté možné propojit s externími účty Google a Microsoft. Množství propojených externích providerů není omezené, tudíž je možné propojit I více různých účtů od jednoho providera.

Princip registrace lokálního uživatele je takový, že po načtení webové aplikace uživatel zvolí „Přihlásit se/Registrovat“ a vyplní požadovaný formulář. V momentě stisknutí tlačítka „Potvrdit“ se provede ověření vstupních hodnot. Registraci uživatele má na starost metoda **LocalReg**, která má tři vstupní proměnné. Těmi jsou uživatelská přezdívka, heslo a heslo pro ověření jak lze vidět na obrázku 6.9




Obr. 6.9: Formulář pro registraci nového uživatele

Následující parametry, které musí vstupní proměnné splňovat:

- Zvolené uživatelské jméno musí mít minimální délku 3 znaky a maximální délku 49 znaků
- Heslo musí mít minimální délku 3 znaky a maximální délku 20 znaků
- Pole „Heslo“ a pole „Zadejte heslo znovu“ musí být shodná

Pokud jedna z těchto podmínek není splněna, registrace neproběhne a uživatel je o tom informován pomocí hlášky „Zadané parametry jsou neplatné.“, po třech vteřinách je automaticky přesměrován zpět do registrační sekce.

V případě vložení správných parametrů proběhne ověření unikátnosti zvolené přezdívky. Toto ověření je provedeno na základě volání databáze, kdy je využita

funkce `Any()`, která vrací hodnotu `true` (v případě nalezení stejné přezdívky v databázi), nebo `false` (pokud přezdívka zatím neexistuje). V případě hodnoty `true` je uživateli zobrazena chybová hláška „Zadanou přezdívku (přezdívka) nelze zvolit. Je již registrovaná.“. Kód pro toto ověření lze vidět na následujícím obrázku 6.10.

```
//Ověření platnosti vstupních proměnných
if ((nick.Length > 2) & (nick.Length < 50) && (password1 == password2) && (password1.Length > 2) & (password1.Length < 21))
{
    //Ověření unikátnosti registrovaného uživatele hledáním v databázi
    bool nalezeno = (from LokalNick in db.LocalAccountsSet where LokalNick.Email.Equals(nick) select LokalNick.Email).Any();
    if (nalezeno == true) //Ověření již existující přezdívky
    {
        //Chybová hláška zobrazená ve View ve webové aplikaci
        ViewBag.result = "Zadanou přezdívku " + nick + " nelze zvolit. Je již registrovaná.";
        //Přesměrování s časovým zpožděním 3 vteřiny na registrační stánku
        Response.AddHeader("REFRESH", "3;URL=http://oauthprojekt.localtest.me/Home/Login");
        return View(); //Zobrazení View ve webovém prohlížeči

    }
```

Obr. 6.10: Ukázka kódu pro ověření vstupních parametrů

Hodnota `false` umožní přistoupení k části pro registraci, kde se uživateli vytvoří nový záznam v databázi v tabulce **LocalAccounts**.

Položky zapsané do tabulky jsou následující:

**LocalID** tato hodnota je náhodně vygenerována. Datovým typem je guid a slouží pro jednotnou identifikaci skrze ostatní tabulky a databázi.
**Email** zde je ve formátu string uložena uživatelova přezdívka.
**Password** zahašované heslo, které uživatel zadal při registraci.

Po zapsání uživatele do tabulky **LocalAccounts** je zavolána metoda `LoginControl`, která slouží pro zapsání uživatele do přihlášených uživatelů. Registrovaný uživatel je tedy přihlášený a přesměrovaný na úvodní stránku webové aplikace. Ukázka kódu pro zapsání nového uživatele do databáze je na následujícím obrázku 6.11

```
//Vytvoření nové proměnné pro zápis záznamu do talubulky LocalAccounts
var LocalAccounts = new LocalAccounts();
//Zašifrování hesla pomocí algoritmu SHA1
string crypted_pass = BitConverter.ToString(sha1.ComputeHash(ASCIIEncoding.Default.GetBytes(password1+nick)));

Guid guid = Guid.NewGuid(); //Vygenerování nového Guid
LocalAccounts.LocalId = guid;
LocalAccounts.Email = nick;
LocalAccounts.Password = crypted_pass;
db.LocalAccountsSet.Add(LocalAccounts); //Zapsání nové položky do databáze
db.SaveChanges(); //Uložení změn v databázi

LoginControl(guid); //Zavolání metody pro přihlášení uživatele
Response.AddHeader("REFRESH", "0;URL=http://oauthprojekt.localtest.me/"); //Přesměrování na úvodní stránku
return View(); //Zobrazení View ve webovém prohlížeči
```

Obr. 6.11: Kód pro zapsání nového uživatele do databáze

Pro přihlášení uživatele, který má existující lokální účet, se postupuje obdobným způsobem jako při registraci. Uživatel zvolí „Přihlásit se/Registrovat“ a vyplní požadovaný formulář. Přihlášení uživatele má na starost metoda `LocalLog`, která má dvě vstupní proměnné. Těmi jsou uživatelské jméno a heslo viz obrázek 6.12.


Obr. 6.12: Formulář pro přihlášení

Následující parametry, které musí vstupní proměnné splňovat:

- vložené uživatelské jméno musí mít minimální délku 3 znaky a maximální délku 49 znaků
- heslo musí mít minimální délku 3 znaky a maximální délku 20 znaků

Pokud jedna z těchto podmínek není splněna, přihlášení neproběhne a uživatel je o tom informován pomocí hlášky „Zadané parametry jsou neplatné“ po třech vteřinách je automaticky přesměrován zpět do registrační sekce.

Pokud jsou výše uvedené parametry splněny, dojde k zahašování hesla a následnému pokusu o nalezení uživatelského jména se zadaným heslem v databázi. Zde je opět jako u registrace využita funkce `Any()`, která vrací hodnotu `true` nebo `false`. Na základě této hodnoty dojde buď k zapsání uživatele do přihlášených pomocí metody **LoginControl**, nebo v opačném případě vypsání chybové hlášky „Bylo zadáno špatné uživatelské jméno či heslo“. Při úspěšném přihlášení je uživatel přesměrován na úvodní stránku, po neúspěšném přihlášení je přesměrován po 3 sekundách opět na registrační stránku. Na následujícím obrázku je znázorněn kód pro lokální přihlášení uživatele 6.13.

```
//Ověření platnosti vstupních proměnných
if ((nick.Length > 2) && (nick.Length < 50) && (password.Length > 2) & (password.Length < 21))
{
    //Zašifrování zadaného hesla
    string crypted_password = BitConverter.ToString(sha1.ComputeHash(ASCIIEncoding.Default.GetBytes(password+nick)));
    //Vyhledání uživatele se zadaným uživatelským jménem a heslem
    bool nalezeno = (from LokalNick in db.LocalAccountsSet where LokalNick.Email.Equals(nick)
                     where LokalNick.Password.Equals(crypted_password) select LokalNick.Email).Any();
    if (nalezeno == true)
    {
        //nalezení jednotného identifikátoru v rámci aplikace
        Guid guid = (from lokalId in db.LocalAccountsSet where lokalId.Email.Equals(nick)
                     where lokalId.Password.Equals(crypted_password) select lokalId.LocalId).First();
        LoginControl(guid); //zavolání metody pro přihlášení

        //zavolání metody pro indikaci přihlášeného uživatele
        ViewBag.user = HomeController.CheckLogin(System.Web.HttpContext.Current);
        //přesměrování uživatele na úvodní stránku
        Response.AddHeader("REFRESH", "0;URL=http://oauthprojekt.localtest.me/");

        return View();
    }
```

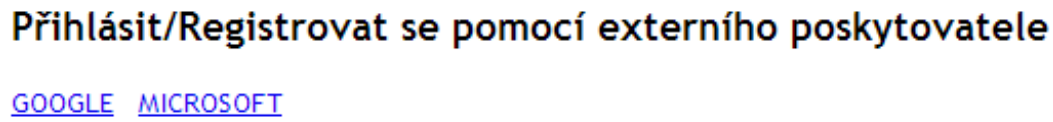
Obr. 6.13: Ukázka kódu pro lokální přihlášení

## 6.4 Registrace a přihlášení externího uživatele

Jednou ze stěžejních funkcí webové aplikace je možnost přihlásit uživatele do aplikace pomocí externího providera. To znamená, že uživatel nemusí vytvářet nový účet a může se přihlásit pomocí stávající služby, která podporuje technologii SSO. V případě této aplikace se jedná o služby Google a Microsoft.

Komunikace mezi klientem (webová aplikace) a providerem (server poskytující autentizaci) probíhá pomocí protokolu OAuth 2.0 u obou služeb.

Mechanismus externího přihlášení je v aplikaci takový, že uživatel ve webové aplikaci zvolí „Přihlásit se/Registrovat“ a v části „Přihlásit/Registrovat se pomocí externího poskytovatele“ uživatel vybere jednoho providera viz obrázek 6.14.




Obr. 6.14: Sekce pro výběr externího přihlášení

Po zvolení providera je uživatel přesměrován na stánku s přihlášením, kde vyplní přihlašovací údaje a odsouhlasí aplikaci. Po potvrzení je odeslána odpověď od providera do metody `LoginOrRegister`, která se nachází ve třídě **ExtLogin**. Metoda `LoginOrRegister` má za úkol zjistit, zda má proběhnout registrace uživatele, nebo přihlášení.

Nejprve je zavolána metoda `ExtLogin`, která má na starost vyměnit získaný autorizační kód za přístupový token, kterým aplikace zažádá API služby o uživatelský

e-mail a o ID uživatele. V metodě je mechanismus, který zjišťuje, zda jde o přihlášení pomocí služby Google, nebo Microsoft. V dalším kroku metody `ExtLogin` je odeslán autorizační kód společně s informacemi o klientovi `client_id secret` providerovi. Poté je odpověď přijata a uložena do proměnné typu string s názvem `responseFromServer`.

Jelikož je odpověď ve formátu JSON, nelze s proměnnou přímo pracovat. Je třeba získat string pouze s hodnotou přístupového tokenu. To je provedeno pomocí metody `JsonConvert`, pomocí níž je možné deserializovat přijatou odpověď, a tím získat pouze potřebnou hodnotu proměnné `access_token`. Mechanismus metody `ExtLogin` se nachází na následujícím obrázku 6.15.

```
//Odeslání požadavku pomocí POST
WebRequest request = WebRequest.Create(client.Url);
request.Method = "POST";
string postData = "code=" + code + "&client_id=" + client.ClientId + "&client_secret=" + client.ClientSecret +
    "&redirect_uri=" + client.RedirectUri + "&grant_type=authorization_code";
byte[] byteArray = Encoding.UTF8.GetBytes(postData);
request.ContentType = "application/x-www-form-urlencoded";
request.ContentLength = byteArray.Length;
Stream dataStream = request.GetRequestStream();
dataStream.Write(byteArray, 0, byteArray.Length);
dataStream.Close();

//Příjmutí odpovědi
WebResponse response = request.GetResponse();
dataStream = response.GetResponseStream();
StreamReader reader = new StreamReader(dataStream);
string responseFromServer = reader.ReadToEnd();
reader.Close();
dataStream.Close();
response.Close();

//Deserializování odpovědi s přístupovým tokenem
var des_accesstoken = JsonConvert.DeserializeObject<dynamic>(responseFromServer);
var accesstoken = des_accesstoken.access_token;
```

Obr. 6.15: Komunikace aplikace pomocí POST s providerem

V tomto momentě je možné přistupovat k API (viz obrázek 6.16) dané služby s přístupovým tokenem, a tím získat požadovanou e-mailovou adresu a ID uživatele. Odpověď je opět ve formátu JSON, tudíž je nutné ji deserializovat.

```
//volani gooogle API s požadavkem na uživatelovi základní informace
WebRequest requestAPI = WebRequest.Create("https://www.googleapis.com/oauth2/v1/userinfo?access_token="
    + accesstoken);
Stream objStream;
objStream = requestAPI.GetResponse().GetResponseStream();
StreamReader objReader = new StreamReader(objStream);
string sLine = objReader.ReadToEnd();

//Deserializace odpovědi
var des_email = JsonConvert.DeserializeObject<dynamic>(sLine);
string email = des_email.email;
string externalId = des_email.id;

return new Tuple<string, string, string>(externalId,email, "google");
```

Obr. 6.16: Volání API Google s přístupovým tokenem

Metoda `ExtLogin` vrací tři proměnné typu string. První proměnnou je ID uživatele u služby, druhou proměnnou je e-mail a v poslední se nachází název služby, pomocí které se provádělo externí přihlášení, či registrace.

Na základě vrácených informací nyní metoda `LoginOrRegister` zjišťuje, zda se uživatel se získaným ID (ID od služby) nachází v databázi, či nikoliv. Toto rozhodování se provádí vyhledáváním v tabulce `ExternalAccounts`. Vyhledání je řešeno metodou `Any()`.

V případě nalezení existujícího uživatele je zavolána metoda `ExternalLogin`, která přihlásí existujícího uživatele. Pokud uživatel neexistuje, tak je spuštěna metoda `ExternalRegister`. Tělo metody `LoginOrRegister` je zobrazeno na následujícím obrázku 6.17.

```
//vyhledávání uživatele podle ID získaného od providera
bool nalezeno = (from ExternalLogin in db.ExternalAccountsSet where ExternalLogin.ServiceId.Contains(ext.Item1)
                 where ExternalLogin.Service.Contains(ext.Item3) select ExternalLogin.ServiceId).Any();

if (nalezeno == true)
{
    //v případě nalezení existujícího uživatele se provede přihlášení
    ExternalLogin(ext.Item1, ext.Item3);
    ViewBag.result = "Uživatel byl úspěšně přihlášen.";
    Response.AddHeader("REFRESH", "2;URL=http://oauthprojekt.localtest.me/");
    return View();
}
else
{
    //v případě nenalezení uživatele se provede registrace
    ExternalRegister(ext.Item1, ext.Item2, ext.Item3);
    ViewBag.result = "Uživatel " + ext.Item2 + " byl úspěšně registrován.";
    Response.AddHeader("REFRESH", "2;URL=http://oauthprojekt.localtest.me/");
    return View();
}
```

Obr. 6.17: Rozhodování mezi přihlášením a registrací

V případě přihlášení se jedná o jednoduchou metodu s názvem `ExternalLogin` (obrázek 6.18). Tato metoda má dvě vstupní proměnné. První vstupní proměnou je externí ID získané od providera a druhou proměnou je typ služby `google, microsoft`. Podle těchto údajů je nalezena v databázi lokální přezdívka uživatele. Poté je zavolána metoda `LoginControl` ze třídy `AccountControlConrtoller`, která slouží k zapsání uživatele do přihlášených uživatelů.

```
public void ExternalLogin(string externalId, string service)
{
    using (var db = new OAuthDataContext())
    {
        //nalezení lokálního uživatele v databázi podle externího ID a typu služby
        var x = (from a in db.ExternalAccountsSet
                 join b in db.LocalAccountsSet on a.LocalAccountsLocalId equals b.LocalId
                 where a.ServiceId.Contains(externalId) where a.Service.Contains(service)
                 select new { Email = b.Email, LocalId = a.LocalAccountsLocalId }).First();


      //zavolání metody LoginControl pro zapsání uživatele do přihlášených uživatelů
      AccountControlController.LoginControl(x.LocalId);
      ViewBag.user = x.Email;
    }
}
```

Obr. 6.18: Metoda pro externí přihlášení

Registrace externího uživatele probíhá pomocí metody `ExternalRegister` (obrázek 6.19), metoda má tři vstupní proměnné. Těmi jsou externí ID, uživatelův e-mail a typ služby, ze které registrace proběhla. Na základě těchto informací je uživateli vygenerován jednotný identifikátor v rámci aplikace (guid). V databázi je do tabulky **LocalAccounts** zapsána uživatelova přezdívka převzatá ze služby (e-mail) a Guid, pole pro heslo zůstává prázdné. Do tabulky **ExternalAccounts** je uloženo externí ID, typ služby a Guid. Databáze je uložena a metoda zavolá metodu `LoginControl` pro zapsání uživatele mezi přihlášené uživatele. Uživatel je tedy po registraci ihned přihlášen.

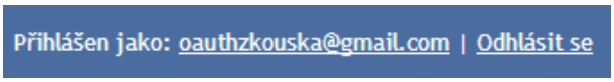
```
public void ExternalRegister(string externalID, string uzivateluvEmail, string service)
{
    using (var db = new OAuthDataContext())
    {
        var LocalAccounts = new LocalAccounts();
        Guid guid = Guid.NewGuid(); //vygenerování nového guid
        LocalAccounts.LocalId = guid;
        LocalAccounts.Email = uzivateluvEmail;

        db.LocalAccountsSet.Add(LocalAccounts); //zapsání informací do tabulky LocalAccounts

        Guid newguid = Guid.NewGuid();
        var ExternalAccounts = new ExternalAccounts();
        ExternalAccounts.Service = service;
        ExternalAccounts.ServiceId = externalID;
        ExternalAccounts.LocalId = newguid;
        ExternalAccounts.LocalAccountsLocalId = guid;
        db.ExternalAccountsSet.Add(ExternalAccounts);
        db.SaveChanges(); //uložení databáze

        //zavolání metody LoginControl pro zapsání uživatele do přihlášených uživatelů
        AccountControlController.LoginControl(guid);

        ViewBag.user = uzivateluvEmail;
    }
```

Obr. 6.19: Metoda pro externí registraci

## 6.5 Přidání externího účtu

Další funkcí aplikace je možnost propojení existujícího lokálního účtu s externím účtem Google a Microsoft. Připojit další externí účty je možné i pro účet registrovaný externě. Např. pokud se uživatel zaregistroval pomocí služby Google, může si připojit i službu Microsoft. Poté bude jedno, zda se přihlásí účtem Google nebo Microsoft, protože bude přihlášen do stejného účtu.

Připojení další služby pro přihlášení je možné tak, že se přihlášený uživatel přepne do svého nastavení profilu. Přepnutí do profilu se provádí kliknutím na přezdívku uživatele v indikační části aplikace dle obrázku 6.20.




Obr. 6.20: Indikační část aplikace

V dolní části stránky u nastavení profilu se nachází možnost připojení další služby. V této části se nachází jednoduchá indikace, zda má uživatel nějakou externí službu propojenou s aplikací. Aplikaci indikuje stavy, kdy uživatel nemá žádného externího poskytovatele. Kdy má jednoho z poskytovatelů nebo jestli jsou registrováni oba poskytovatelé. Není zde indikace množství propojených účtů od jedné služby. Jedná se tedy pouze o obecnou informaci pro uživatele, nikoliv o přesný výčet. Na výběr jsou opět obě služby i v případě, že je uživatel již přes jednu službu registrován. Toto řešení je zvoleno pro případ, že by uživatel měl více účtů na jedné službě a chtěl by je mít propojené.

V momentě zvolení jedné ze služeb proběhle opět přesměrování na daného providera, kde se uživatel přihlásí a povolí přístup aplikaci. Po odsouhlasení je uživatel navrácen zpět na stránku aplikace - tímto je propojení úspěšně dokončené.

Mechanismus propojení je realizovaný metodou `LoginOrRegister`, kde jsou hlavní kroky ověření, zda je uživatel stále přihlášený a zda účet, který se má propojit, není již propojený s jiným účtem. V případě splnění těchto podmínek (obrázek 6.21) dojde k připojení externího uživatelského účtu.

```
string user = HomeController.CheckLogin(System.Web.HttpContext.Current);
if (user.Contains("no_user")) //podmínka přihlášení uživatele
{
    //chybová hláška v případě odhlášeného uživatele
    ViewBag.result = "Uživatel je odhlášený. Přihlašte se prosím znovu.";
    Response.AddHeader("REFRESH", "3;URL=http://oauthprojekt.localtest.me/Home/Login");
    return View();
}

//dotaz na databázi, zda účet, který se má propojit již existuje v databázi
bool exist = (from extrenalMail in db.ExternalAccountsSet where extrenalMail.ServiceId.Contains(ext.Item1)
              select extrenalMail.ServiceId).Any();

if (exist == true) //rozhodování v případě existence externího účtu v databázi
{
    //v případě existujícího propojení se zobrazí chybová hláška
    ViewBag.result = "Zadaný účet " + ext.Item2 + " nelze propojit s lokálním účtem. Je již registrován";
    Response.AddHeader("REFRESH", "3;URL=http://oauthprojekt.localtest.me/Home/Login");
    return View();
}
```

Obr. 6.21: Podmínky pro připojení účtu

## 6.6 Přihlášení a indikace přihlášeného uživatele

Princip přihlášení uživatele je následující: pro přihlášení je zavolána metoda **LoginControl**. Tato metoda má jednu vstupní proměnnou `localId` datového typu guid. Této metodě je předáno guid uživatele, který se úspěšně autentizoval (lokálně, či pomocí externího providera).

Metoda **LoginControl** přistupuje do databáze k tabulce **LoginControl**, která slouží k ukládání přihlášených uživatelů. Do tabulky se zapíše proměnná `localId` (guid uživatele), je zapsán čas vytvoření záznamu v tabulce do položky `Created` a je vygenerována položka `token`, která je typu guid.

Dále je v rámci metody **LoginControl** zavolána metoda **WriteCookie**, která zapíše do cookies (s názvem `LoginControl`) uživatelova prohlížeče vygenerovaný token. Tímto je proces přihlášení kompletní.

Součástí aplikace je i jednoduchá indikace přihlášeného uživatele. Zda je uživatel přihlášený, je indikováno v horním pravém rohu webové aplikace. Pokud je uživatel přihlášen, je zobrazena jeho přezdívka a možnost okamžitého odhlášení.

Indikace, zda je uživatel přihlášený, probíhá pomocí metody **CheckLogin**. Metoda si nejprve vyžádá cookies uživatelova prohlížeče s názvem **LoginControl**. V případě, že cookies neexistují, je indikován odhlášený uživatel.

Pokud se cookies v prohlížeči nachází, tak je obsah cookies porovnáván s proměnnou token v databázi (tabulka LoginControl). V případě, že jsou oba tokeny shodné, provede se dle proměnné `localId` vyhledání uživatelské přezdívky, a ta je poté vrácena metodou.

Pokud uživatel není ověřen, ať už z důvodu neexistujícího cookies či neshodného tokenu, tak metoda vrací string `no_user`. Na základě této informace lze nepřihlášenému uživateli omezit přístup do některých částí webové aplikace, jako je například vstup do „Soukromá sekce“.

Existuje zde možné riziko v odcizení tokenu uloženého v cookies prohlížeče. V případě odcizení by se mohl útočník vydávat za přihlášeného uživatele. Tento problém je z části vyřešen omezenou časovou platností tokenu. Token má životnost 30 minut v cookies prohlížeče a v databázi je uložen po dobu 5 minut. Uživatel je tedy přihlášen pouze 5 minut, poté se musí přihlásit znovu.

Promazávání databáze má na starost metoda `Application_BeginRequest`, která je spuštěna vždy při nové žádosti poslané aplikaci. Promazávání spočívá v porovnání proměnné `created` (čas vytvořený v databázi) s aktuálním časem. Pokud je čas delší než 5 minut, je údaj vymazán.

Z hlediska výše uvedených rizik toto řešení indikace není zcela bezpečné, pro testovací účely aplikace však dostačuje.

## 6.7 Testování a výsledky

V praktické části byla realizována webová aplikace, jež je funkční a protokol OAuth funguje bez problémů. Aplikace byla otestována na operačních systémech Windows 7 a Windows 8 v prostředí Microsoft Visual Studio Express 2012 for Web.

Jedinou komplikací, která při realizaci nastala, bylo testování aplikace lokálně. Provider Microsoft neumožňuje zasílat odpovědi na localhost `127.0.0.1`. Adresa pro odpověď (redirect URI) musí být reálná doména. Tento problém byl vyřešen pomocí služby localtest.me, která přesměruje příchozí provoz na localhost.

Problémem však je, že localtest.me nepodporuje protokol HTTPs. Z tohoto důvodu jsou odpovědi od providera ke klientské aplikaci v HTTP, a tudíž je možné je odposlechnout. Komunikace, která je ve směru od klientské aplikace k providerovi, je šifrovaná, tudíž nehrozí vyzrazení identifikačních údajů aplikace.

Tento problém nastává pouze v případě testování na localhost. V momentě umístění aplikace na reálnou doménu tento problém odpadá a je možné použít protokol HTTPs v obou směrech.

# 7 ZÁVĚR

Úkolem této bakalářské práce bylo popsat systémy jednotného přihlášení (SSO). V úvodní části byly zmíněny nezbytné teoretické základy autentizace a autorizace, následované způsoby komunikace, jež tyto systémy využívají, tzn. metody GET a POST.

V práci jsou popsány dva v současné době nejpoužívanější standardy pro realizaci systémů SSO. Jedná se o standardy OpenID a OAuth, u nichž obou je podrobně popsán princip fungování a průběhu při ověřování identity. Dále následuje porovnání těchto standardů.

Bylo zjištěno, že oba standardy mají své specifické využití. Protokol OpenID je vhodnější pro autentizaci uživatele pomocí libovolného OpenID providera. Naopak u standardu OAuth je aplikace svázána s určitým providerem. OAuth poskytuje však kromě autentizace také autorizaci a snazší možnost přístupu k uživatelským datům. Ačkoli má OpenID nespornou výhodu v možnosti použití libovolného providera, byla zvolena metoda OAuth pro svou funkci autorizace a možnost přístupu k datům dle úrovně autorizace.

V následující části se práce zabývá praktickou realizací protokolu OAuth ve webové aplikaci. Byl popsán způsob registrace aplikace u jednotlivých providerů a procesy nutné k úspěšnému ověření. Je uveden příklad přístupu k API providera u ověřeného uživatele.

Poslední část práce popisuje program, který byl vytvořen dle zadání bakalářské práce. Byla vytvořena webová aplikace v ASP.NET, kde si uživatel může zvolit externí přihlášení pomocí providera Google či Misrosoft. V aplikaci je možné vytvořit lokální uživatelský účet, který lze později propojit s externím účtem. K účtu vytvořenému pomocí externího přihlášení je taktéž možné přidat lokální heslo. Realizovaná aplikace je plně funkční.

Během testování bylo zjištěno, že aplikace funguje s Oauth dle očekávání spolehlivě. Standard OAuth je tedy vhodný k realizaci aplikace s možností jednotného přihlášení pomocí služeb Google a Microsoft.

Aplikaci by bylo možné do budoucna rozšířit o možnost implementace více providerů, či otestovat offline mód u protokolu OAuth, který lze využít pro klasické desktopové aplikace.

# LITERATURA


[1] FFIEC. *Authentication in an Internet Banking Environment.* 2011. Dostupné z: <http://www.digitallibrary.kcci.com.pk/bitstream/32417747/701/1/authentication_guidance.pdf>.

[2] KRHOVJÁK, Jan a Václav MATYÁŠ. Autentizace a identifikace uživatelu. *Zpravodaj ÚVT MU* [online]. 2007 [cit. 2013-05-18]. Dostupné z: <http://www.ics.muni.cz/bulletin/articles/560.html#lit1>.

[3] KORPELA, Jukka. Methods GET and POST in HTML forms - what's the difference?. *IT and communication* [online]. 2001, 2003-09-28 [cit. 2013-05-18]. Dostupné z: <http://www.cs.tut.fi/~jkorpela/forms/methods.html>.

[4] HTTP/1.1: Method Definitions. *World Wide Web Consortium (W3C)* [online]. 1999 [cit. 2013-05-18]. Dostupné z: <http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9>.

[5] MIESSLER, Daniel. URLs vs. URIs: Differences and Examples. *Danielmiessler.com* [online]. [cit. 2013-05-18]. Dostupné z: <http://danielmiessler.com/study/url_vs_uri/>.

[6] RFC 3986 - Uniform Resource Identifier (URI): Generic Syntax. IETF. *The Internet Engineering Task Force (IETF)* [online]. 2005 [cit. 2013-05-18]. Dostupné z: <http://tools.ietf.org/html/rfc3986>.

[7] OpenID Authentication 2.0 - Final. *OpenID Foundation website* [online]. 2007 [cit. 2013-05-19]. Dostupné z: <http://openid.net/specs/openid-authentication-2_0.html>.

[8] The OAuth 2.0 Authorization Framework. *The Internet Engineering Task Force* [online]. 2012 [cit. 2013-05-27]. Dostupné z: <http://tools.ietf.org/html/draft-ietf-oauth-v2-31#section-1.3.1>.

[9] MACDONALD, Matthew. *Beginning ASP.NET 4.5 in C#.* New York: Distributed to the book trade worldwide by Springer Science Business Media New York, c2012, xxxvi, 885 p. Expert's voice in .NET. ISBN 14-302-4251-5.

[10] Úvod do architektury MVC. *Zdroják* [online]. 2009 [cit. 2013-05-20]. Dostupné z: <http://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/>.

[11] *ScottGu's Blog* [online]. 2003 [cit. 2013-05-20]. Dostupné z: <http://weblogs.asp.net/scottgu/>.

[12] Introducing LocalDB, an improved SQL Express. *MSDN Blogs* [online]. 2011 [cit. 2013-05-20]. Dostupné z:<http://blogs.msdn.com/b/sqlexpress/archive/2011/07/12/introducing-localdb-a-better-sql-express.aspx>.

[13] Autentizace pomoci Google OAuth 2.0. *Imp.cz* [online]. 2012 [cit. 2013-05-20]. Dostupné z: <http://blog.imp.cz/post/2012/05/28/Autentizace-pomoci-Google-OAuth-20>.

[14] Using OAuth 2.0 for Web Server Applications. *Google Developers* [online]. 2013 [cit. 2013-05-22]. Dostupné z: <https://developers.google.com/accounts/docs/OAuth2WebServer>.

# SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

API programovací aplikační rozhraní - Application Programing Interface

ASCII americký standardní kód pro výměnu informací - American Standard Code for Information Interchange

ASP rozhraní pro programování aplikací - Active Server Pages

HTTP hypertextový protokol - Hypertext Transfer Protocol

HTTP zabezpečený hypertextový protokol - Hypertext Transfer Protocol Secure

IIS internetová informační služba - Internet Information Server

JSON JavaScript Object Notation

MVC Model-View-Controller

SHA hašovací algoritmus - Secure Hash Algorithm

SQL strukturovaný dotazovací jazyk - Structured Query Language

SSL zabezpečený přenos - Secure Sockets Layer

SSO systém jednotného přihlášení - Single Sign-On

UI uživatelské rozhraní - User Interface

URI jednotný identifikátor zdroje - Uniform Resource Identifier

URL jednotný identifikátor cesty - Uniform Resource Locator

URN jednotný identifikátor názvu - Uniform Resource Name

USB univerzální sériová sběrnice - Universal Serial Bus

# SEZNAM PŘÍLOH

# A OBSAH ELEKTRONICKÉ PŘÍLOHY

Na přiloženém DVD se nachází elektronická verze bakalářské práce ve formátu PDF. Dále se zde nachází složka s kompletním projektem Visual studia. Adresářová struktura důležitých součástí přílohy je následující:

- Složka `aplikace` obsahující projekt Visual studia.
- bakalářská práce - název souboru `bakalarska_prace_MatousPlasil.pdf`
- Soubor `readme.txt` s nápovědou